Streaming Data management and Time Series Analysis project

University of Milano-Bicocca

Matteo Corona - 838138

Contents

List of Figures

List of Tables

1 Preliminary steps

1.1 Importing libraries

First of all all the necessary libraries have been imported.

```
library(ggplot2)
library(gridExtra)
library(RColorBrewer)
library(lubridate)
library(ggpubr)
library(knitr)
library(kableExtra)
library(dplyr)
library(tseries)
library(xts)
library(forecast)
library(KFAS)
library(Metrics)
```

1.2 Loading the dataset

The first step was to load the dataset from the *.csv* file. I also checked for any missing value but the dataset was complete.

```
# Importing the dataset
data <- read.table("data2022_train.csv", header = TRUE, sep = ",")
# Checking for missing values
if (any(is.na(data))) {
  cat("Sono presenti valori mancanti\n")
} else {
  cat("Nessun valore mancante\n")
}
```

```
## Nessun valore mancante
```

```
# Changing column names for clarity
colnames(data) <- c(
  "time",
  "power")
```

2 Exploratory data analysis

Exploratory Data Analysis (EDA) is a critical initial step in the data analysis process. In this section, an EDA is conducted on the dataset in order to gain a deeper understanding of its characteristics. The attributes in the dataset are:

- **time**: string encoding the date-time of the measurement, in the format dd/mm/yyyy HH:MM:SS The data covers the period from 01/01/2017 00:00:00 - 30/11/2017 23:50:00 and it is sampled every 10 minutes.
- **power**: detected electricity consumption

First of all it is necessary to convert the *time* column into a *datetime* format. There were problems usign the *as.POSIXct()* function which didn't recognize the timestamps. These problems were solved by imposing the timezone to UTC. For time series analysis it is also necessary to convert the data into a *time series* object using the *ts()* method. I also divided the dataset into two parts: training set and test set, keeping the last month for making comparison between model predictions and actual values.

```r
# Converting the time_column to a datetime format
data$time <- as.POSIXct(data$time, format = "%Y-%m-%d %H:%M:%S",
                        tz = "UTC", xts_check_TZ = FALSE)
# Setting the number of observations to keep for the test set
num_test_observations <- 4320
# Spliting the data into training and test sets
train_data <- data[1:(nrow(data) - num_test_observations), ]
test_data <- data[(nrow(data) - num_test_observations + 1):nrow(data), ]

# Converting data to a time series object
ts <- ts(train_data$power, train_data$time)
```

The dataset has 48096 total observations: 43776 will be used for the train set and 4320 for the test set. Let's now start our analysis by simply printing the first samples of the dataset in order to explore the available data.

```r
# Printing the first five observations
kable(data[1:5,], booktabs = T, caption = "Structure of the dataset") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 1: Structure of the dataset

time	power
2017-01-01 00:00:00	34055.70
2017-01-01 00:10:00	29814.68
2017-01-01 00:20:00	29128.10
2017-01-01 00:30:00	28228.86
2017-01-01 00:40:00	27335.70

```r
# Checking the structure of the dataset
dim(data)
```

```
## [1] 48096     2
```

It is useful to plot some summary statistics about the *power* attribute.

```r
# Calculating summary statistics for the 'value' column
power_min <- min(data$power)
power_q1 <- quantile(data$power, probs = 0.25)
power_median <- median(data$power)
power_mean <- mean(data$power)
power_q3 <- quantile(data$power, probs = 0.75)
power_max <- max(data$power)

# Creating a data frame to store the summary statistics
power_summary_df <- data.frame(
  statistic = c("Minimum", "1st Quartile", "Median", "Mean", "3rd Quartile", "Maximum"),
  value = c(power_min, power_q1, power_median, power_mean, power_q3, power_max)
)
# Printing the 'power' summary
kable(power_summary_df, booktabs = T, caption = "Power consumption statistics") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 2: Power consumption statistics

statistic	value
Minimum	13895.70
1st Quartile	26612.98
Median	32650.50
Mean	32643.25
3rd Quartile	37579.85
Maximum	52204.40

Figure 1 shows the distribution of the electricity consumption. In this distribution plot, three main regions appear to have significantly higher frequency, indicating a multimodal distrubution of the electricity consumption variable.

```
# Plotting distribution of electricity consumption using ggplot2
ggplot(data, aes(x = power)) +
  geom_histogram(binwidth = 1, na.rm = TRUE, color = "darkgrey") +
  scale_x_continuous(limits = c(17000, 53000)) +
  scale_y_continuous(limits = c(0, 13)) +
  labs(x = "Electricity Consumption", y = "Frequency",
       title = "Distribution of Electricity Consumption")
```

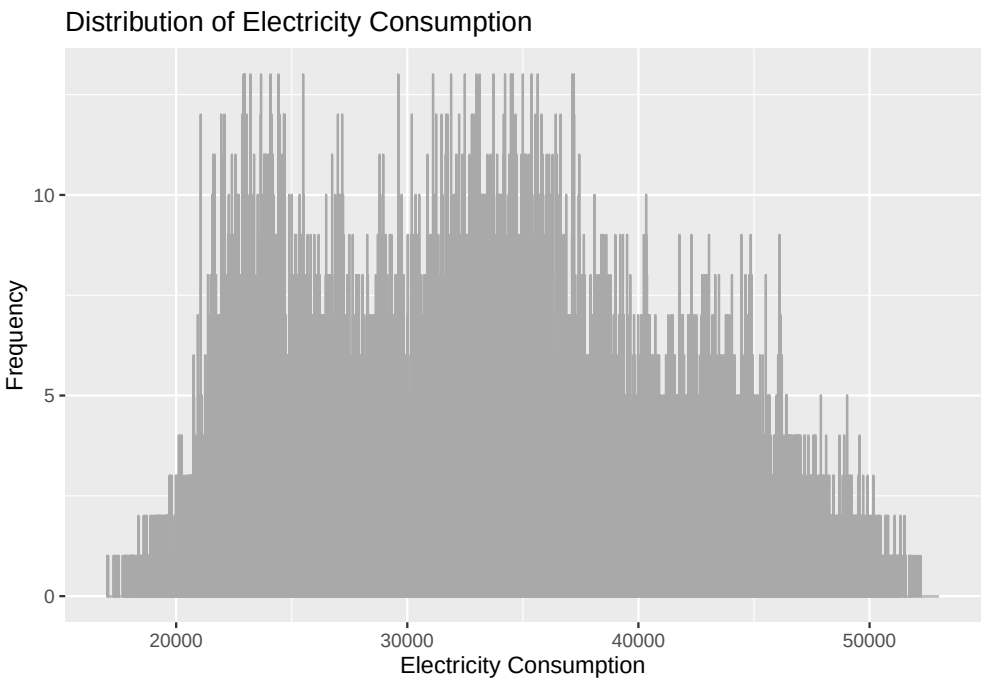


Figure 1: Distribution of electicity consumption

Finally it is of course very useful to plot and analyze the data as a time series. Figure 2 shows the time series three times: in the first plot the time series is shown in its integrity, then it is sampled every 8 hours (3 times a day) and finally it is sampled daily. This plot allow to begin visualizing the pattern and the trends in the data. For example it can be noticed that power consumption have higher values during warm months.

```r
xts <- xts(data$power, data$time)
# Creating a data frame from the 'ts' object for original data
df_original <- data.frame(date = time(xts), power = as.vector(xts))

# Creating a beautiful time series plot for the original data
plot_original <- ggplot(data = df_original, aes(x = date, y = power)) +
  geom_line(color = "#0072B2", size = 1) +
  labs(title = "Original Power Time Series", x = NULL, y = "Power") +
  theme_minimal() +
  theme(plot.title = element_text(size = 20, face = "bold", hjust = 0.5),
        axis.title.x = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1),
        axis.text.y = element_text(size = 12),
        legend.position = "none",
        panel.grid.major = element_line(color = "#E5E5E5"),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"),
        plot.background = element_rect(fill = "#F5F5F5"))

# Creating a plot sampled every 8 hours (3 times a day)
three_times_a_day_data <- xts[endpoints(xts, on = "hours", k = 8), ]
df_three_times_a_day <- data.frame(date = time(three_times_a_day_data),
                                   power = as.vector(three_times_a_day_data))

plot_three_times_a_day <- ggplot(data = df_three_times_a_day, aes(x = date, y = power)) +
  geom_line(color = "#0072B2", size = 1) +
  labs(title = "Data Sampled 3 Times a Day", x = NULL, y = "Power") +
  theme_minimal() +
  theme(plot.title = element_text(size = 20, face = "bold", hjust = 0.5),
        axis.title.x = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1),
        axis.text.y = element_text(size = 12),
        legend.position = "none",
        panel.grid.major = element_line(color = "#E5E5E5"),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"),
        plot.background = element_rect(fill = "#F5F5F5"))

# Creating a daily plot
daily_data <- to.daily(xts, OHLC = FALSE)
df_daily <- data.frame(date = index(daily_data), power = as.vector(daily_data))

plot_daily <- ggplot(data = df_daily, aes(x = date, y = power)) +
  geom_line(color = "#0072B2", size = 1) +
  labs(title = "Daily Average Power Time Series", x = NULL, y = "Power") +
  theme_minimal() +
  theme(plot.title = element_text(size = 20, face = "bold", hjust = 0.5),
        axis.title.x = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1),
        axis.text.y = element_text(size = 12),
        legend.position = "none",
        panel.grid.major = element_line(color = "#E5E5E5"),
        panel.grid.minor = element_blank(),
```

```
        panel.background = element_rect(fill = "white"),
        plot.background = element_rect(fill = "#F5F5F5"))

# Arrange the three plots in columns
grid.arrange(plot_original, plot_three_times_a_day, plot_daily, ncol = 1)
```

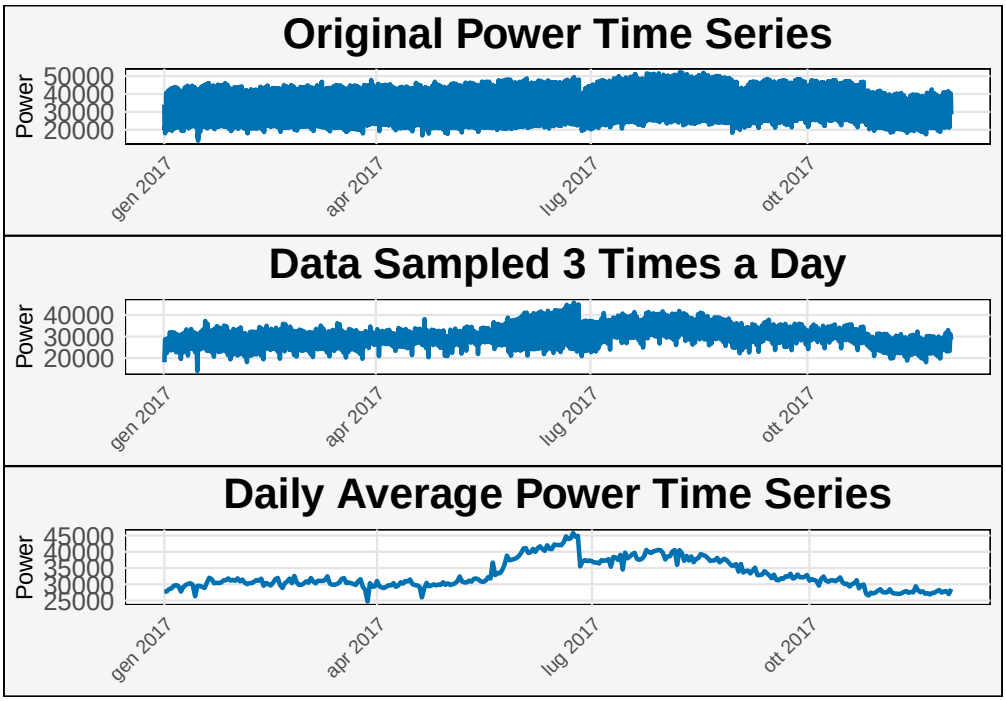


Figure 2: Time series plot

3 Model development

After conducting the Exploratory Data Analysis (EDA), the next step is to proceed with model development. Three different type of models have been considered for forecasting the time series: ARIMA, UCM, and machine learning models.

3.1 ARIMA

The ARIMA (AutoRegressive Integrated Moving Average) model is a popular time series forecasting method. It combines autoregressive (AR) and moving average (MA) components and incorporates differencing to make a non-stationary time series stationary. Let's start the ARIMA analysis by plotting correlograms. The fact that the ACF (figure 3) has an apparently sinusoidal pattern suggests that there are periodic regularities in the data, implying the possible presence of a seasonal or cyclic pattern in the time series. On the other hand, figure 4 shows the PACF and it can be seen that PACF has significant value for the first 5 lags.

```
# Computing and plotting acf with confidence intervals
acf_result <- acf(ts, lag = 144, main = "Autocorrelation Function",
                  xlab = "Lag", ylab = "ACF Value")
grid()
```

```
# Computing and plotting pacf with confidence intervals
pacf_result <- pacf(ts, lag = 144, main = "Autocorrelation Function",
```

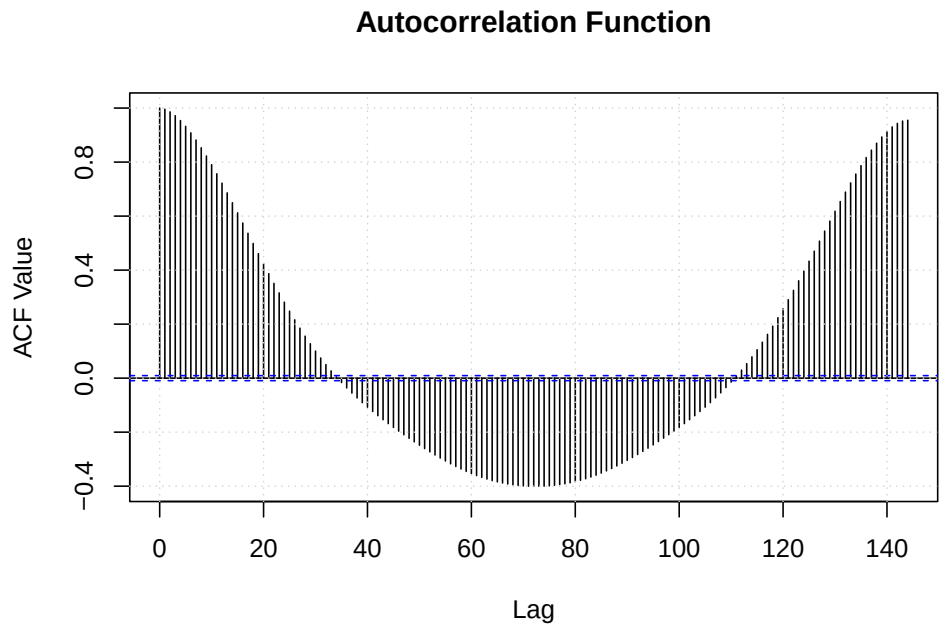


Figure 3: ACF

```
                    xlab = "Lag", ylab = "ACF Value")
grid()
```

The ACF in figure 3 suggest that the series is not stationary. Since the ACF has peaks every 144 lags (one day), it is necessary to differentiate the series in order to make it stationary. Figure 5 shows the daily differentiated time series that looks stationary in mean and variance.

```
# Creating the differentiated time series
ts_diff <- diff(ts, 144)

# Creating a data frame for ggplot2
df <- data.frame(Time = seq_along(ts_diff), Difference = ts_diff)

# Creating the plot using ggplot2
ggplot(df, aes(x = Time, y = Difference)) +
  geom_line(color = "blue", size = 1) +
  labs(
    title = "Daily Differentiated Time Series",
    x = "",
    y = "Power"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),  # Center the title
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank()
  )
```

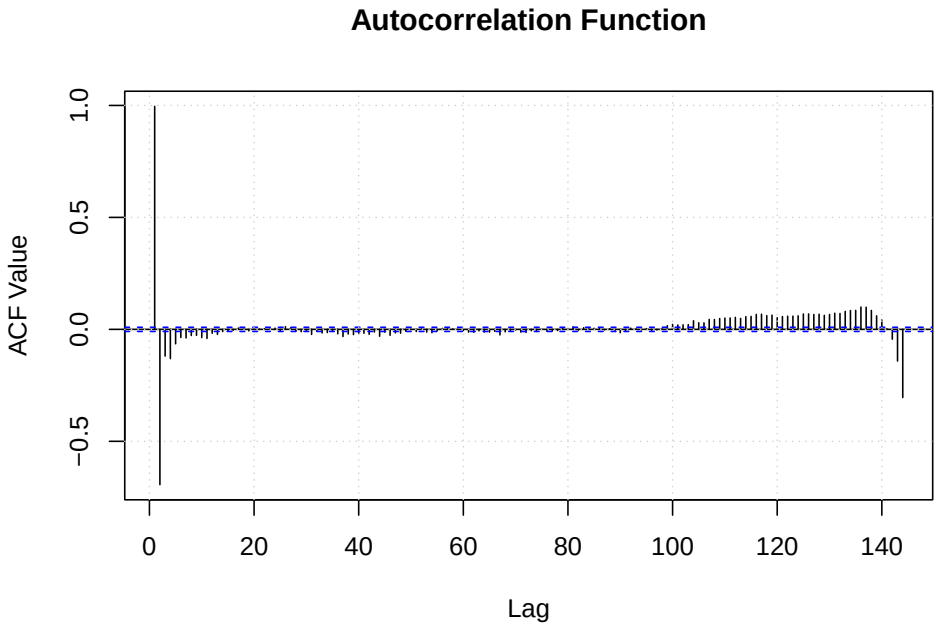


Figure 4: PACF

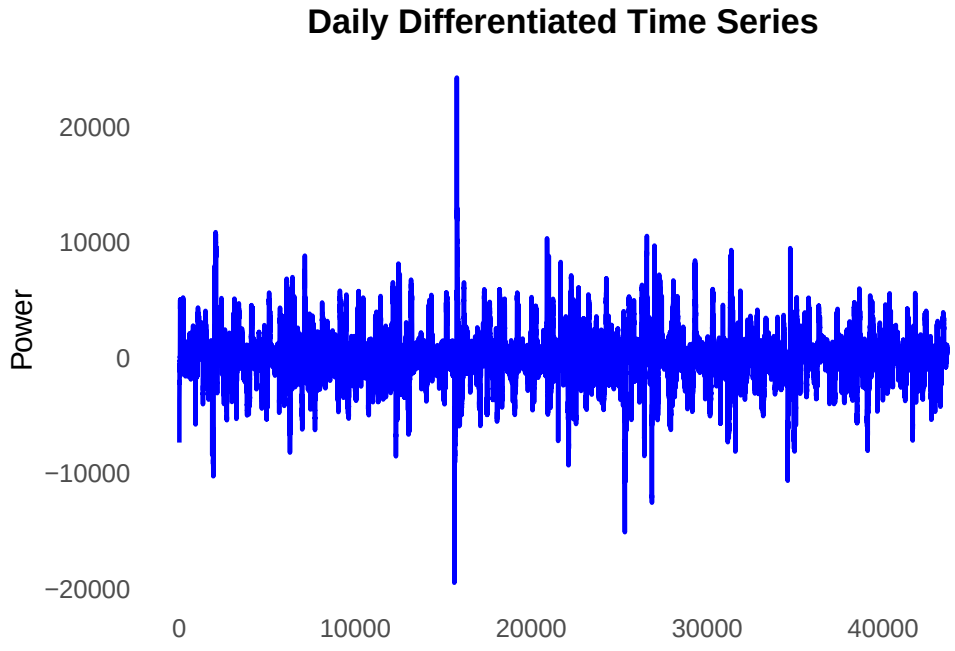


Figure 5: Daily differentiated time series

An *Augmented Dickey-Fuller Test* has been computed in order to check if the differentiated series is stationary or not. The obtained p-value is 0.01, indicating that the differentiated series can be considered stationary based on the assumptions of this test.

```
# Performing the Augmented Dickey-Fuller test
adf_test <- adf.test(ts_diff)
```

```
## Warning in adf.test(ts_diff): p-value smaller than printed p-value
```

```
# Printing the test results
print(adf_test)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_diff
## Dickey-Fuller = -19.489, Lag order = 35, p-value = 0.01
## alternative hypothesis: stationary
```

Let's now try to compute a first ARIMA model. Since the data has daily seasonality, 144 will be included as a parameter for the non-seasonal part of ARIMA. Looking at ACF and PACF in figure 3 and 4 the decision of implementing an ARIMA(4,0,0)(0,1,0)[144] was made.

```
# Computing and saving ARIMA(4,0,0)(0,1,0)[144] model
arima_1 <- Arima(ts, order = c(4, 0, 0), seasonal = list(order = c(0, 1, 0), period = 144))
```

After having trained the model it is possible to compute predictions and make comparison with the test dataset. Figure 6 show the ARIMA(4,0,0)(0,1,0)[144] predictions compared to the test dataset.

```
# Defining the number of future time points you want to predict
n_pred <- 4320

# Generating a forecast object for future time points
forecast_arima_1 <- forecast(arima_1, h = n_pred)

# Combining the true values and predictions into a single dataframe
results_arima_1 <- data.frame(True_Values = test_data$power,
                              ARIMA_Predictions = forecast_arima_1$mean)
```

```
# Creating a time series plot with custom x-axis
ggplot(results_arima_1, aes(x = seq(ymd_hm("2017-11-01 00:00"),
                                    by = "10 mins", length.out = length(True_Values)))) +
  geom_line(aes(y = True_Values, color = "True Values")) +
  geom_line(aes(y = ARIMA_Predictions, color = "ARIMA Predictions")) +
  labs(x = "Time", y = "Power") +
  scale_color_manual(values = c("True Values" = "blue", "ARIMA Predictions" = "red")) +
  theme_minimal()
```

Let's now compute some metrics in order to evaluate the model.

```
# Computing and saving metrics (MAE MAPE RMSE)
true_values <- results_arima_1$True_Values
predictions_arima_1 <- results_arima_1$ARIMA_Predictions
mae_value_arima_1 <- mae(true_values, predictions_arima_1)
rmse_value_arima_1 <- rmse(true_values, predictions_arima_1)
mape_value_arima_1 <- mean(abs((true_values - predictions_arima_1) / true_values)) * 100
# Creating a data frame for the metrics
metrics_df <- data.frame(
```

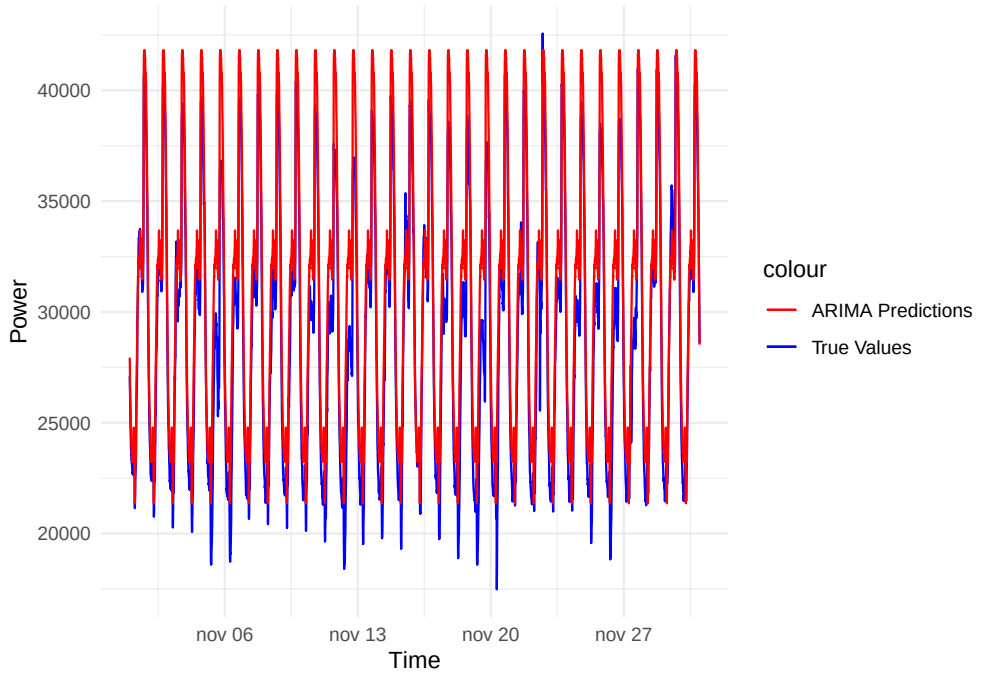


Figure 6: ARIMA(4,00)(0,1,0)[144] predictions

```
  Metric = c("RMSE", "MAE", "MAPE"),
  Value = c(rmse_value_arima_1, mae_value_arima_1, mape_value_arima_1)
)
# Creating a kable table for the metrics
kable(metrics_df,align = "c", caption = "ARIMA(4,0,0)(0,1,0)[144] evaluation metrics") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 3: ARIMA(4,0,0)(0,1,0)[144] evaluation metrics

Metric	Value
RMSE	2109.281662
MAE	1772.183749
MAPE	6.380277

Since the obtained RMSE and MAE for the ARIMA(4,0,0)(0,1,0)[144] model are quite high, the model has been improved by adding sinusoidal regressor.

```
# Defining sinusoidal regressors
frequency_yearly <- 2 * pi * (1 / (365 * 24 * 6))
train_data$sin_yearly <- sin(frequency_yearly * 1:length(train_data$time))
train_data$cos_yearly <- cos(frequency_yearly * 1:length(train_data$time))
# Creating a matrix with sinusoidal regressors
xreg_matrix <- cbind(train_data$sin_yearly, train_data$cos_yearly)
# ARIMA model with sinusoidal regressors
arima_2 <- Arima(ts, order = c(4, 0, 0),
                 seasonal = list(order = c(0, 1, 0),period = 144),
                 xreg = xreg_matrix)
```

```r
# Creating the sinusoidal regressors for test_data
test_data$sin_yearly <- sin(frequency_yearly * 1:length(test_data$time))
test_data$cos_yearly <- cos(frequency_yearly * 1:length(test_data$time))

# Creating a matrix with the sinusoidal regressors for test_data
xreg_matrix_test <- cbind(test_data$sin_yearly, test_data$cos_yearly)
# Generating a forecast object for future time points with exogenous regressors
forecast_arima_2 <- forecast(arima_2, h = n_pred, xreg = xreg_matrix_test)

# Combining the true values and predictions into a single dataframe
results_arima_2 <- data.frame(True_Values = test_data$power,
                              ARIMA_Predictions = forecast_arima_2$mean)
```

Let's visualize the predictions for the ARIMA(4,0,0)(0,1,0)[144] model with sinusoidal regressors and then compute metrics for the test dataset.

```r
# Creating a time series plot with custom x-axis
ggplot(results_arima_2, aes(x = seq(ymd_hm("2017-11-01 00:00"),
                                    by = "10 mins", length.out = length(True_Values)))) +
  geom_line(aes(y = True_Values, color = "True Values")) +
  geom_line(aes(y = ARIMA_Predictions, color = "ARIMA Predictions")) +
  labs(x = "Time", y = "Power") +
  scale_color_manual(values = c("True Values" = "blue", "ARIMA Predictions" = "red")) +
  theme_minimal()
```

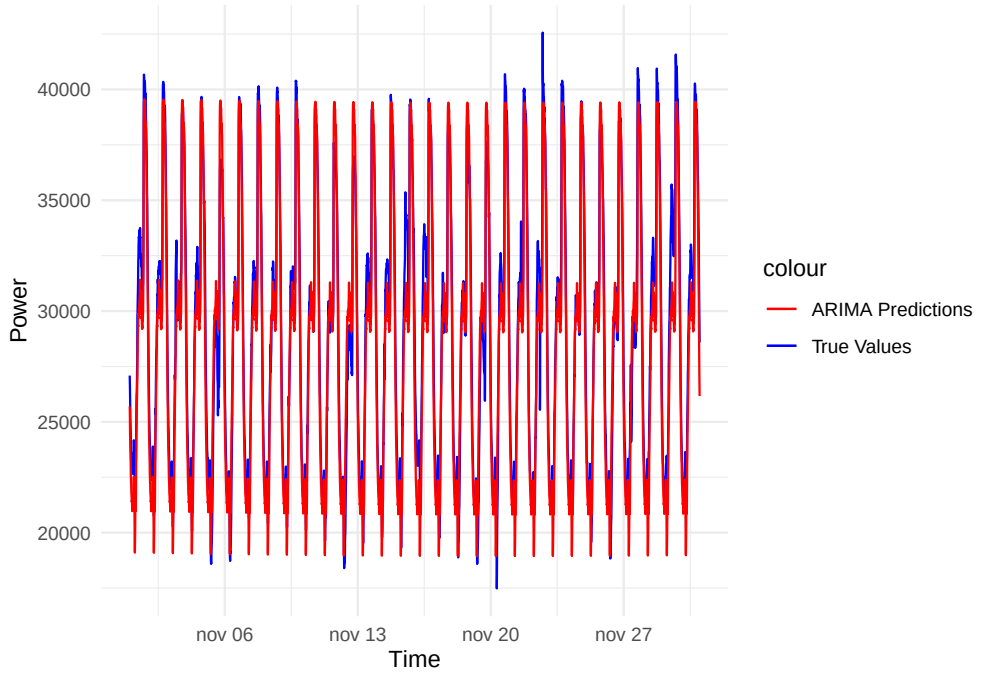


Figure 7: ARIMA(4,00)(0,1,0)[144] (+ regressors) predictions

```r
# Computing and saving metrics (MAE MAPE RMSE)
predictions_arima_2 <- results_arima_2$ARIMA_Predictions
mae_value_arima_2 <- mae(true_values, predictions_arima_2)
rmse_value_arima_2 <- rmse(true_values, predictions_arima_2)
mape_value_arima_2 <- mean(abs((true_values - predictions_arima_2) / true_values)) * 100
```

```r
# Creating a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("RMSE", "MAE", "MAPE"),
  Value = c(rmse_value_arima_2, mae_value_arima_2, mape_value_arima_2)
)
# Creating a kable table for the metrics
kable(metrics_df,align = "c", caption = "ARIMA(4,0,0)(0,1,0)[144] evaluation metrics") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 4: ARIMA(4,0,0)(0,1,0)[144] evaluation metrics

Metric	Value
RMSE	1505.288616
MAE	1219.176445
MAPE	4.357324

3.2 UCM

Let's now continue the analysis by implementing a UCM model. UCM (Unobserved Components Models) are time series models that decompose a time series into various unobserved components, each representing a different source of variation. These components typically include level, trend, seasonality, and error terms. UCM models are used for time series forecasting and understanding the underlying patterns in the data by separating them into these distinct components. Among UCM models, the SSM has been chosen. SSM (State Space Model) is a framework for modeling time series data, separating hidden states from observed measurements. In the definition of the SSM model, RW (Random Walk) has been used. The model incorporates both trend and seasonal components to capture underlying patterns in the data. It uses the variance, denoted as 'vy', computed from the training data to set the initial state covariance matrix and parameter values, ensuring that the model adapts to the data characteristics.

```r
# Calculating vy using your time series data
vy <- var(train_data$power, na.rm = TRUE)

# Defining UCM model with vy included
ucm_1 <- SSModel(power ~ 0 +
                 SSMtrend(1, NA) +
                 SSMseasonal(144, 0, 'trigonometric', harmonics = 1:10),
                 H = NA,
                 data = train_data
)

# Setting the initial covariance matrix using vy
diag(ucm_1$P1inf) <- 0
diag(ucm_1$P1) <- vy

# Defining parameter values (pars) using vy
pars <- log(c(
  logVarEta  = vy/10,
  logVarZeta = vy/10,
  logVarOm7  = vy/100,
  logVarOm365= vy/100,
  logVarEps  = vy/10
))
```

```r
# Fiting the UCM model with the defined parameters
ucm_fit_1 <- fitSSM(ucm_1, pars)
```

Figure 8 shows predictions for the implemented UCM model and then evaluation metrics on the test dataset are presented.

```r
# Generating a forecast object for future time points
forecast_ucm_1 <- predict(ucm_fit_1$model, n.ahead = n_pred)

# Combining the true values and predictions into a single dataframe
results_ucm_1 <- data.frame(True_Values = test_data$power,
                            UCM_Predictions = as.vector(forecast_ucm_1))

# Creating a time series plot with custom x-axis
ggplot(results_ucm_1, aes(x = seq(ymd_hm("2017-11-01 00:00"),
                                  by = "10 mins", length.out = length(True_Values)))) +
  geom_line(aes(y = True_Values, color = "True Values")) +
  geom_line(aes(y = UCM_Predictions, color = "UCM Predictions")) +
  labs(x = "Time", y = "Power") +
  scale_color_manual(values = c("True Values" = "blue", "UCM Predictions" = "red")) +
  theme_minimal()
```

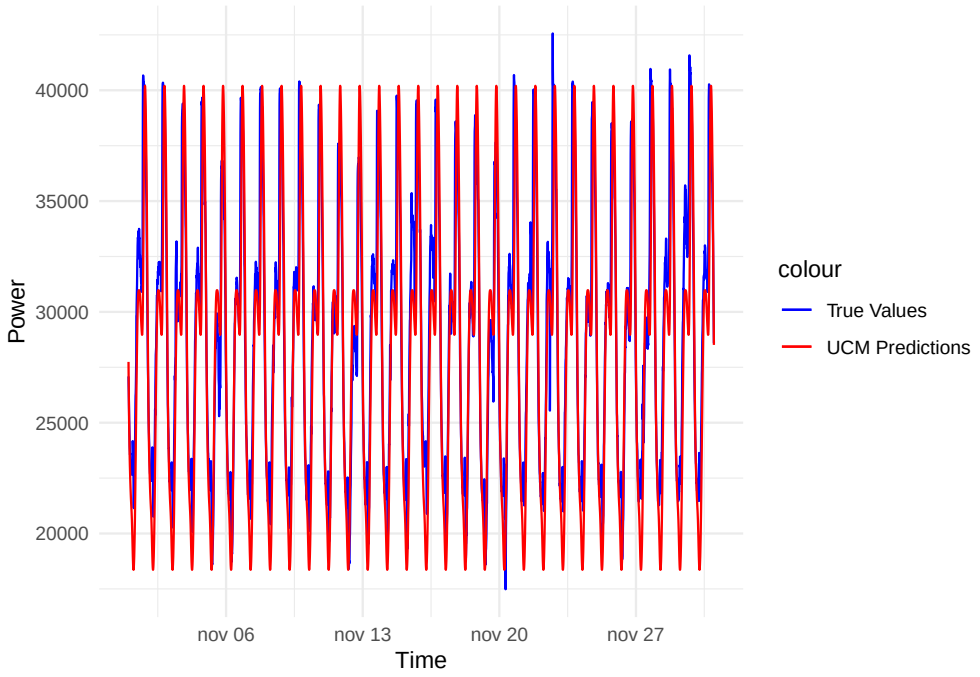


Figure 8: UCM predictions

```r
predictions_ucm_1 <- results_ucm_1$UCM_Predictions
# Calculating MAE (Mean Absolute Error)
mae_value_ucm_1 <- mae(true_values, predictions_ucm_1)
rmse_value_ucm_1 <- rmse(true_values, predictions_ucm_1)
mape_value_ucm_1 <- mean(abs((true_values - predictions_ucm_1) / true_values)) * 100
# Creating a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("RMSE", "MAE", "MAPE"),
```

```
  Value = c(rmse_value_ucm_1, mae_value_ucm_1, mape_value_ucm_1)
)
# Creating a kable table for the metrics
kable(metrics_df,align = "c", caption = "UCM evaluation metrics") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 5: UCM evaluation metrics

Metric	Value
RMSE	2571.214780
MAE	1899.091384
MAPE	6.441264

3.3 Machine Learning

After implementing ARIMA and UCM time series forecasting models, machine learning models were tested. These models were implemented in Python. Different models were implemented:

- RandomForestRegressor
- LinearRegression
- GradientBoostingRegressor
- DecisionTreeRegressor

In order for the Machine Learning models to be trained, I decided to extract some features from the time column. The following table show the time feature which were selected. The 'holidays' feature was added exploiting the *holidays* library in Python.

```
ml_start <- read.table("ML_start.csv", header = TRUE, sep = ",")
kable(ml_start[1:5,], booktabs = T, caption = "Dataset for ML models") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 6: Dataset for ML models

X	Year	Month	Weekday	Hour	Quarter	Day	Day_of_year	Weekend	holidays	power
0	2017	1	6	0	1	1	1	1	1	34055.70
1	2017	1	6	0	1	1	1	1	1	29814.68
2	2017	1	6	0	1	1	1	1	1	29128.10
3	2017	1	6	0	1	1	1	1	1	28228.86
4	2017	1	6	0	1	1	1	1	1	27335.70

The complete python code can be seen here. The following table shows the evaluation metric for the two Machine Learning models.

```
# Creating a data frame with the metrics
metrics_df <- data.frame(
  Model = c("RandomForestRegressor", "LinearRegression",
            "GradientBoostingRegressor", "DecisionTreeRegressor"),
  RMSE = c(1147, 4281, 2090, 1155),
  MAE = c(731, 3848, 1127, 734),
  MAPE = c(2.28, 12.28, 4.80, 2.29)
)
```

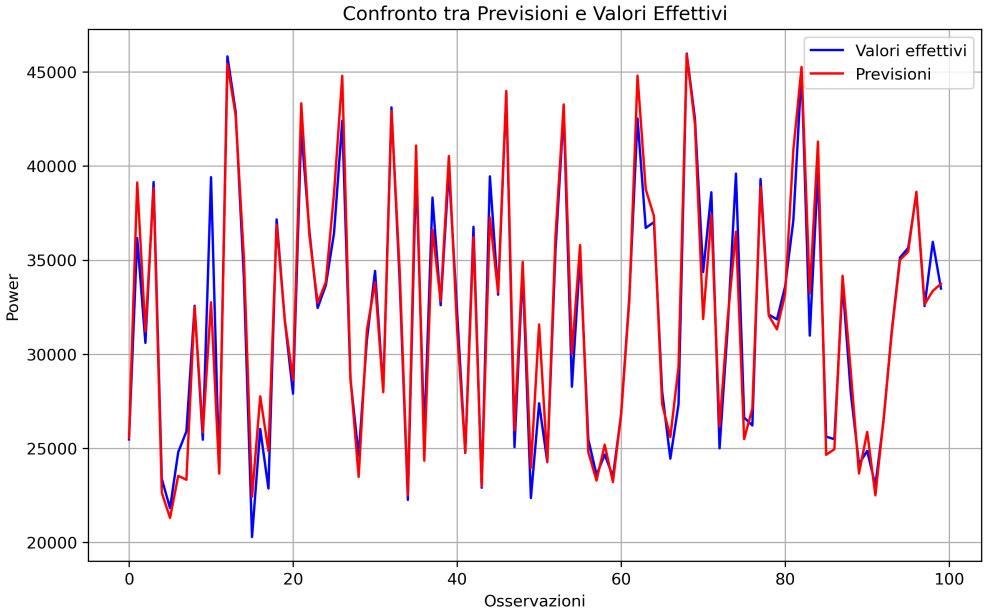
```r
# Creating a kable table with styling
metrics_df %>%
  kable(align = "c", caption = "ML evaluation metrics") %>%
  kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 7: ML evaluation metrics

Model	RMSE	MAE	MAPE
RandomForestRegressor	1147	731	2.28
LinearRegression	4281	3848	12.28
GradientBoostingRegressor	2090	1127	4.80
DecisionTreeRegressor	1155	734	2.29

The best model seems to be the RandomForestRegressor, which has been chosen for predictions on December future values. For the purpose of training the models, the dataset was splitted into two parts, leaving 38476 elements for the training and the remaining 9620 for the testing. The following image shows some of the predicion on the test set along with the corresponding original values.

```r
knitr::include_graphics("ML_predictions.png")
```



In order to compute the december predictions with the selected Machine Learning model, a new dataset with december timestamps was created:

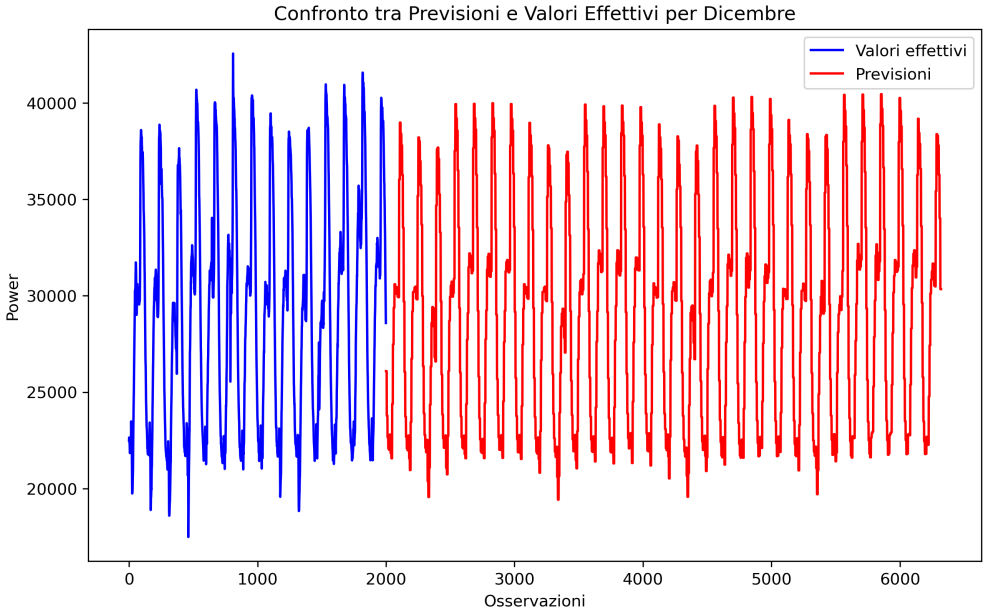
```r
ml_december <- read.table("ML_december.csv", header = TRUE, sep = ",")
kable(ml_december[1:5,], booktabs = T, caption = "Dataset for ML december predictions") %>%
kable_styling(latex_options = c("striped", "HOLD_position"))
```

Table 8: Dataset for ML december predictions

X	Year	Month	Weekday	Hour	Quarter	Day	Day_of_year	Weekend	holidays
0	2017	12	4	0	4	1	335	0	0
1	2017	12	4	0	4	1	335	0	0
2	2017	12	4	0	4	1	335	0	0
3	2017	12	4	0	4	1	335	0	0
4	2017	12	4	0	4	1	335	0	0

The following image shows the December prediction right after the last entries of the original dataset.

```
knitr::include_graphics("ML_predictions_december.png")
```



4 Conclusions

In conclusion, for the task of electricity consumption prediction, several models were tested, including ARIMA models with sinusoidal regressors, UCM, and many machine learning models. Among all the models, the machine learning models, particularly the RandomForestRegressor model, achieved superior performance in terms of evaluation metrics. Consequently, based on the results, the RandomForestRegressor model emerged as the most effective approach for accurately forecasting electricity consumption.